CaQR: A Compiler-Assisted Approach for Qubit Reuse through Dynamic Circuit

Fei Hua
huafei90@gmail.com
Rutgers University
USA

Yuwei Jin
yj243@scarletmail.rutgers.edu
Rutgers University
USA

Yanhao Chen
chenyh64@gmail.com
Rutgers University
USA

Suhas Vittal
suhaskvittal@gmail.com
Georgia Institute of Technology
USA

Kevin Krsulich
kevin.krsulich@ibm.com
IBM T. J. Watson Research Center
USA

Lev S. Bishop
lsbishop@us.ibm.com
IBM T. J. Watson Research Center
USA

John Lapeyre
john.lapeyre@ibm.com
IBM T. J. Watson Research Center
USA

Ali Javadi-Abhari
ali.javadi@ibm.com
IBM T. J. Watson Research Center
USA

Eddy Z. Zhang
eddy.zhengzhang@gmail.com
Rutgers University
USA

ABSTRACT

Quantum measurement is important to quantum computing as it extracts out the outcome of the circuit at the end of the computation. Previously, all measurements have to be done at the end of the circuit. Otherwise, it will incur significant errors. But it is not the case now. Recently IBM starts supporting dynamic circuit through hardware (instead of software by simulator). With mid-circuit hardware measurement, we can improve circuit eficacy and fidelity from three aspects: (a) reduced qubit usage, (b) reduced swap insertion, and (c) improved fidelity. We demonstrate this using real-world applications Bernstein Verizani on real hardware and show that circuit resource usage can be improved by 60%, and circuit fidelity can be improved by 15%. We design a compiler-assisted tool that can find and exploit the tradeoff between qubit reuse, fidelity, gate count, and circuit duration. We also developed a method for identifying whether qubit reuse will be beneficial for a given application. We evaluated our method on a representative set of important ap-plications. We can reduce resource usage by up to 80% and improve circuit fidelity by up to 20%.

CCS CONCEPTS

• Computer systems organization → Quantum computing.

KEYWORDS

qubit reuse,mid-circuit measurement, circuit fidelity, qubit usage

ACM Reference Format:
Fei Hua, Yuwei Jin, Yanhao Chen, Suhas Vittal, Kevin Krsulich, Lev S. Bishop, John Lapeyre, Ali Javadi-Abhari, and Eddy Z. Zhang. 2023. CaQR: A Compiler-Assisted Approach for Qubit Reuse through Dynamic Circuit. In Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3 (ASPLOS '23), March 25–29, 2023, Vancouver, BC, Canada. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3582016.3582030

1 INTRODUCTION

Quantum computation is important as it can solve classical intractable problems such as factoring [23] [23], chemistry simulation [12, 21], and large database search [10]. Due to the spectacular advances in quantum hardware, quantum systems have undertaken significant improvement in the past two decades. Now domain experts can run small-scale experiments on real machines for their specific domain problems.

Quantum measurement is at the very heart of quantum computing. It allows classical systems to extract information from the quantum realm. By allowing repeated executions, measurement can gather information of the final state of a qubit in the form of a discrete probability distribution. Previously on IBM Q systems, the measurement is done at the end of a program, for all qubits [28].

Recently IBM started providing hardware support for mid-circuit measurement, as the very first step for supporting dynamic circuit [11]. It has improved measurement gate duration and fidelity on its Falcon family processors [3, 9].

Mid-circuit measurement performed when circuit execution is in flight is very useful. It has two types of functionalities: (1) boolean test of state and (2) qubit reuse. For the boolean test of state, it can be used for stabilizer measurement for quantum error correction code [2] to tell if there is an error in the state. It can also be used for asserting certain properties of a qubit for post-processing purposes. Further, it can be used for steering the computation in a useful direction, for instance, the repeat-until-success (RUS) implementation for synthesizing an arbitrary single-qubit rotation gate [18].

We focus on the functionality of qubit reuse enabled by mid-circuit measurement. In this case, mid-circuit measurement must be combined with mid-circuit qubit reset, which allows the users to reset the qubit state to the ground state at any point of the program

ASPLOS '23, March 25–29, 2023, Vancouver, BC, Canada

ACM ISBN 978-1-4503-9918-0/23/03. . . $15.00
https://doi.org/10.1145/3582016.3582030

execution. Mid-circuit reset is also supported by IBM hardware recently. After a qubit is reset, it can be reused for any other qubit which hasn't started any operation. This capability of saving qubits is important since today's quantum computer size is in the range of 100 qubits. Being able to eficiently reuse the qubits can increase the capacity of the hardware system to run programs.

We show a real application in which qubit count can be significantly reduced by mid-circuit measurement. It is the Bernstein Vazirani (BV) algorithm in Fig. 1. The original circuit uses 5 qubits, but in fact, we can use as few as 2 qubits. We start with the 1-qubit saving case, where for qubit q1, we can perform measure-and-reset after its last gate Hadamard, as shown in Fig. 1 (b). Immediately we let q1 be reused for the gates originally applied to qubit q2. We repeat the same process to reuse q1 for q3, and q4 until we cannot reuse anymore, which results in a 2-qubit circuit, as shown in Fig. 1 (c). In this case, it has reduced as much as 60% usage of qubit resource. Interestingly, for a □-qubit BV application, the minimal number of required qubits is always 2, despite how many qubits are in the original circuit.

Motivated by the resource-saving benefit of qubit reuse, we develop a compiler-assisted approach for automatically transforming circuits exploiting the mid-circuit measurement and reset functionality. Finding proper qubit reuse is challenging. We must also identify the set of applications that can benefit from qubit reuse. With our compiler-assisted tool, users do not have to manually specify whether to reuse or what qubits and when to reuse. Our compiler-assisted tool helps users run large quantum programs on small quantum computers.

Prior studies [5, 19, 27] also optimize qubit usage, but our work is orthogonal to these studies. CutQC [27] spatially distributes the workload of a quantum circuit at a cost of worst-case exponential time classical post-processing. Ancilla qubit reuse [5, 19] requires un-computation. The Square framework [5] explores the tradeoff between un-computation cost and qubit saving. However, our qubit reuse through mid-circuit measurement does not require un-computation and explores a different tradeoff. And in our setting, one qubit can be reused by any other qubit (whether ancilla or non-ancilla) as long as two conditions are satisfied. Therefore the ancilla qubit reuse technique cannot be applied to our problem in this paper.

To our best knowledge, our work is the first to automatically identify qubit-reuse opportunities in general applications. If our tool has identified any qubit-reuse opportunity, it can perform circuit transformation to exploit the trade-off between circuit duration, resource usage, and circuit fidelity. Our tool can handle two different types of applications: the ones with non-commuting gates, and the ones with commuting gates. Furthermore, our tool can also be tuned towards different purposes, towards more qubit saving, or towards gate-count reduction and improved fidelity.

To summarize, our contributions are as follows:

- We discovered non-trivial potential for qubit reuse in real quantum applications through extensive experimentation.
- Our compiler-assisted tool is able to identify if there is any qubit reuse opportunity. If there is any, our tool can yield transformed circuit with respect to different qubit budget level, optimized for circuit duration and fidelity.
- We explore the full spectrum of factors that affect the tradeoff between qubit-reuse and compiled circuit eficiency. We discover that in addition to the benefit of reduced qubit count, qubit-reuse can potentially reduce SWAP insertion and improve circuit fidelity. Hence we designed two versions of our tool such that one emphasizes qubit saving and the other emphasizes SWAP reduction and fidelity.
- Our tool can handle gate commutativity, which is an important feature in modern quantum applications.
- Our experiments show that we can reduce qubit usage by up to 80% while keeping circuit duration similar – slightly larger than the non-reused version by 9.9% on average.
- We also provide experiment results on a real quantum machine IBM Mumbai. We perform experiments on both regular circuit without commuting gates and applications with commuting gates such as QAOA. In both bases, our results show better performance. TVD is improved by 17%. The success rate of finding correct answer increased by 20%. QAOA can converge faster and find better minimal energy. Note that they are under the condition of using fewer qubits.

The rest of the paper is organized as the following. We introduce the background and motivation of qubit reuse in Section 2. We describe the two versions of our compiler-assisted qubit reuse in Section 3. We provide comprehensive experiment evaluation in Section 4. Section 5 describes all related works. Section 6 concludes the paper.

2 BACKGROUND AND MOTIVATION

2.1 Hardware Support for Dynamic Circuit

Recently IBM started providing the dynamic circuit support [11]. In dynamic circuit, it supports mid-circuit measurement operation and mid-circuit reset operation as the example shown in Fig. 2 (a). The measurement operation reads out the qubit. The reset operation forces the state of the measured qubit back to the ground state.

We have made improvements to this combination of measurement and reset. If a qubit is measured as |1> in the standard computation basis, and if we want to reuse this qubit, we must re-initialize it to |0>. If a qubit is measured as |0>, we do not do anything. However, the built-in reset operation denoted as a box containing |0> implicitly contains measurement pulses, which is redundant. So instead of using a combination of a measurement + a reset, we use a measurement + a classical/quantum control not. For the classical/quantum control gate, we use the classical bit to control the quantum bit, as shown in Fig. 2 (b). We can reduce the duration by around 50% with this optimization, from 33.179 dt to 16.467 dt, in IBM Mumbai machine. 1 dt is 0.22 nano-seconds.

To improve readability, we use two vertical bars to represent the combination of mid-circuit measurement and conditional reset, as shown Fig. 1 (a) for the rest of the paper.

2.2 Potential of Qubit Saving

To evaluate the potential of qubit reuse for modern quantum applications, we developed a tool that can automatically generate transformed circuit with (near-)minimal depth/duration for any qubit reuse count, if such reuse is possible. We have created the

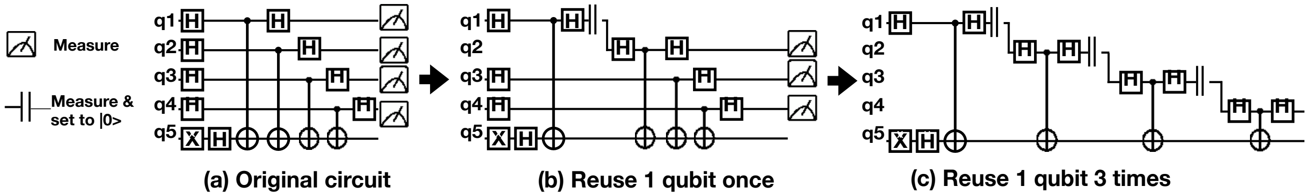


Figure 1: Using Dynamic Circuit Support for the BV Application to Reduce Qubit Usage. (a) Original logical circuit with 5 qubits; (b) Reusing q1 for q2 results in 4 qubits in total; (d) Reusing q1 for q2, q3, and q4 results in 2 qubits in total usage.

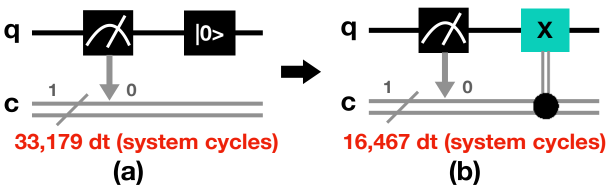
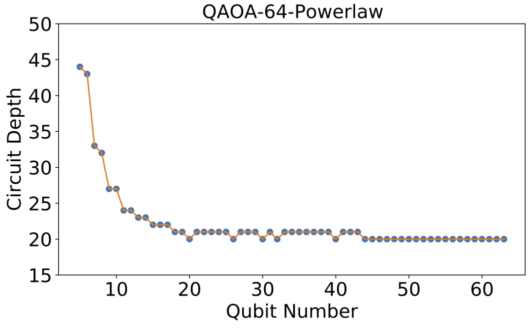
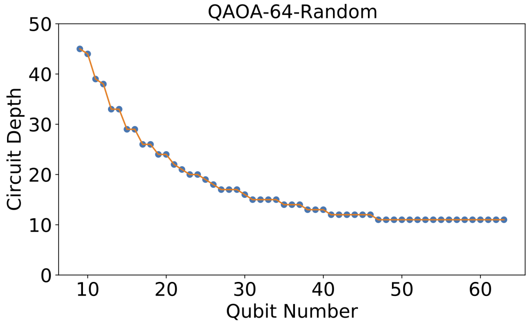


Figure 2: Our improvement for "measurement + reset". (a) Built-in measurement and reset operations in Qiskit; (b) Measurement + classical control which takes half of the time of (a);



(a) Input as a power-law graph with density 30%.



(b) Input as a random problem with density 30%.

Figure 3: The Potential of Qubit Saving by Qubit Reuse

data points with all qubit counts. The design of our tool is described in details in Section 3.2.

We discovered that there are significant opportunities for qubit reuse in real applications. We show experimental results of the QAOA application with 64 qubits for two different input problem graphs – the power law graph and the random graph, both with a density of 30%. It can be seen that qubit usage can be reduced from 64 to as few as 5, which is a significant saving in terms of resources.

It also indicates that as the qubit number decreases, the depth of the circuit increases. However, both cases show (near) heavy-tail distribution, implying that we can potentially reduce qubit usage significantly only by increasing the circuit depth by a reasonably small amount. For the power law graph input to the QAOA program, we can save over 80% qubit while only increasing the circuit duration by at most 25%, as shown in Fig. 3. For the random graph input, we can save 33% qubit by increasing the circuit duration by at most 20%.

2.3 Tradeoffs for Exploiting Dynamic Circuit

After demonstrating the large potential of qubit saving by qubit reuse, now we discuss different factors that affect the final circuit performance/fidelity. As aforementioned, increased qubit reuse may increase the depth (or duration if real gate time is available) of a circuit. However, we discover that there are other benefits brought by qubit reuse, which can potentially offset the disadvantage of increased depth. In a nutshell, by selectively using qubits with higher fidelity and more reliable physical links, we can potentially reduce the number of qubits needed while improving the fidelity and performance of the transformed circuit. We list all benefits of qubit reuse: (1) Qubit Saving, (2) Reduced SWAPs, and (3) Improved Fidelity, and describe each below.

Qubit Saving. We have already described this benefit of reducing qubit usage in Section 1 and Section 2.2.

SWAP Reduction. In addition to the benefit of reducing qubit usage, we can also reduce the number of swaps through qubit reuse. Assuming that the hardware is a 5-qubit physical coupling graph, as shown in Fig. 4 (a).

Note the qubit interaction graph for the 5-qubit BV circuit is a star graph, where q5 has a degree of 4 and all other nodes have a degree of 1, as shown in Fig. 4 (b). However, the maximal degree of a physical qubit is 3. Therefore the 5-qubit logical circuit needs to add SWAPs before it is hardware-compliant.

In contrast, the 4-qubit BV circuit with 1 qubit reuse does not need to have any SWAP inserted if properly mapped to hardware. The qubit-interaction graph's maximal degree is 3, as shown in Fig. 4 (c), by sharing one qubit for q1 and q2. This qubit interaction

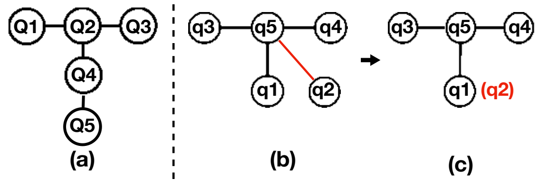


Figure 4: Compiling BV circuit for real architecture. (a) Physical architecture; (b) The qubit interaction graph(if two qubits have a gate, there is an edge) corresponding to the 5-qubit circuit in Fig. 1 (a); (c) The qubit interaction corresponding to the 4-qubit BV circuit in Fig. 1 (b), where q1 is reused by q2. It can be seen that (c) can fit into the physical architecture while (b) cannot.

graph happens to be isomorphic to the hardware architecture in Fig. 4 (a). No SWAP is needed.

Even though the 4-qubit logical BV circuit has a larger depth than the 5-qubit logical BV circuit before SWAP insertion, the final circuit after the hardware mapping stage may end up having a similar duration because of the additional SWAP(s) inserted. A comparison is shown in Fig. 5 (b) and (c).

The reason is that by reusing qubits as if merging multiple nodes into one in the qubit interaction graph, we can potentially alleviate the pressure on hardware connectivity imposed by the original logical circuit.

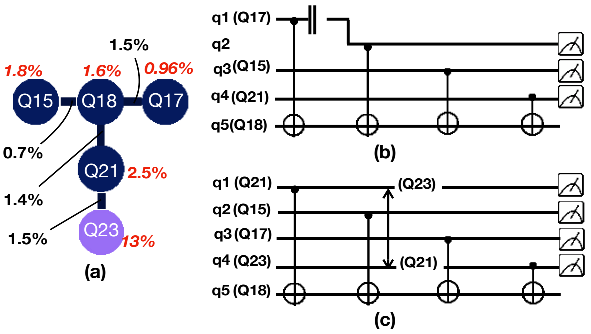


Figure 5: Tackling error variability. For illustration purpose, we eliminate the drawing of Hadamard gates in the figure.

Improved Fidelity. Qubit reuse happens to have the side benefit of improving the fidelity of the compiled circuit. Today's quantum hardware is presented with the challenge of error variability. Some single-qubit gates or two-qubit gates may have larger error rate than others. By reusing qubits, One strategy to improve the overall performance of a quantum system is to selectively exclude qubits with lower fidelity or physical links with higher error rates.

We still use the same example of BV. We map it to the real IBM machine qubits Q15, Q17, Q18, Q21, Q23 on Mumbai. The error rates are listed in Fig. 5 (a). It can be seen that the qubit Q23 has a much higher read out error (13%) than any other qubit (≤ 2.5%). By mapping the 4-qubit BV to only qubits of Q15, Q17, Q18, and Q21, as shown in Fig. 5 (b), we can completely avoid the high readout error on qubit Q23. Since the duration of the two circuits are similar as we measured, the overall circuit of 4-qubit BV is actually better.

We run this experiment on Mumbai to compare the 5-qubit BV (with a SWAP inserted) with the 4-qubit BV circuit in Fig. 6. It shows that the 4-qubit BV is better than 5-qubit BV, in terms of the probability of finding the correct answers, 58% versus 55%. Note that in addition to higher fidelity, we also saved 20% qubit usage.

Interestingly, we find that using 3 qubits is even better (the circuit duration does not increase significantly due to the dependence relationship in the particular circuit). 3-qubit BV does not use SWAP either. We use Q15, Q17, and Q18, since Q15/17/18 all have better readout error rates than Q21, and the link of Q17-Q18 error is similar to that of Q21-Q18. Now the probability of finding the correct answer is improved to 64%, from 55% originally, with 40% qubit saving. In fact, we also tried the 2-qubit BV, but since the circuit duration increase is much larger in 2-qubit BV due to extra measure-reset operations, the 2-qubit BV is not as good as the 3-qubit BV. So we skip the histogram of 2-qubit BV in Fig. 6.

Discussion. To summarize, reusing qubits with dynamic support is useful in that it can save resource usage, reduce SWAP gates, and also potentially improve fidelity. The main disadvantage is increased circuit duration. If the benefits can offset the disadvantage, then qubit-reuse is worth it. It requires a careful analysis to determine which qubits and when to reuse by weighing in all these factors.

We take these factors into consideration in Section 3. We provide two versions of our design: one can save qubits precisely to user demand, and also allows a tradeoff-based tuning, and the other one primarily improves SWAP insertion.

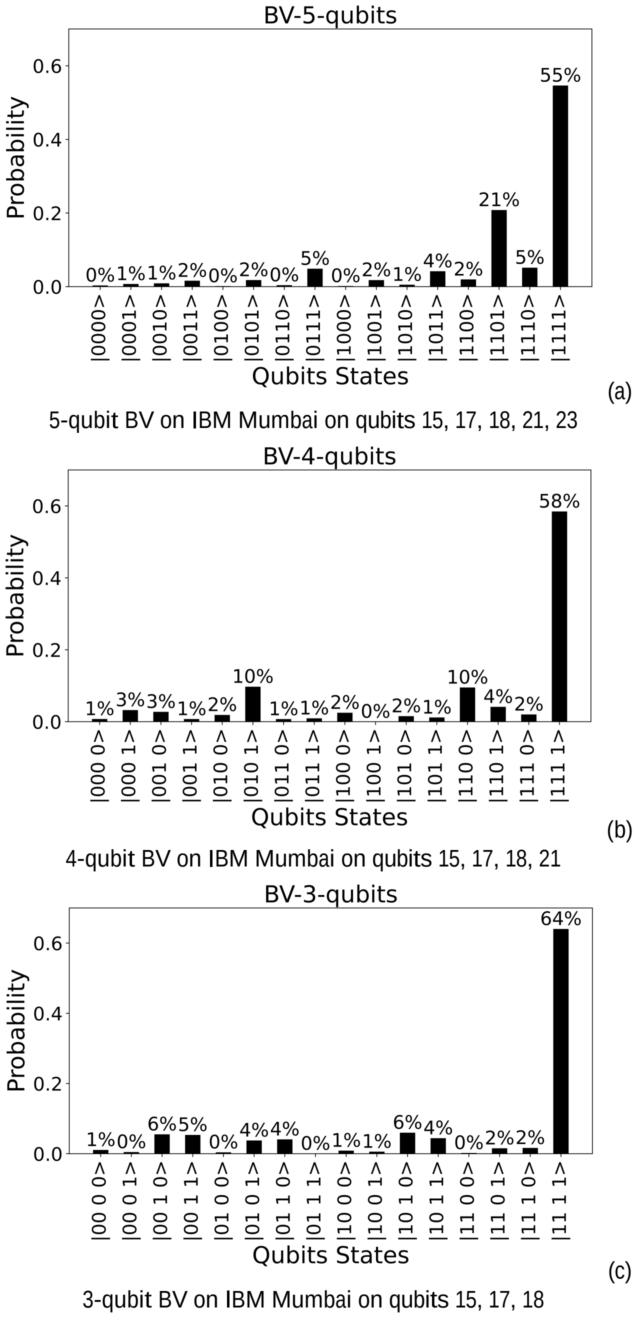
3 DESIGN AND IMPLEMENTATION

In our design, we provide two different versions of compiler-assisted qubit reuse (CaQR). In the first version, we allow users to precisely control the amount of qubit-saving. We generate a transformed and optimized circuit with respect to a given qubit count if there is any. We can also exploit a tradeoff between qubit count, duration (or depth), and fidelity if a range of qubit-saving amount is allowed. We name it qubit saving CaQR (QS-CaQR).

To precisely control the number of qubits in a circuit, we must perform qubit-reuse transformation at the logical circuit level. It is because after the hardware mapping stage, the inserted SWAPs may reduce the opportunities where we can save qubits. After the qubit-reuse transformation is done at the logical circuit level, we perform hardware mapping. The QS-CaQR version is described in Section 3.2.

In the second version, we primarily optimize SWAP-reduction through qubit-reuse. We name it SR-CaQR. For this scenario, the resource is not a problem, i.e., there are enough qubits to implement a circuit. But we want to minimize the gate count and in the meantime, we want to mitigate the impact of error variability in the architecture. Hence, we perform dynamic-circuit aware SWAP insertion and save qubit as a side effect. This version is described in Section 3.3.

Before describing either of the two versions, we first formally define the conditions when a qubit could be reused.



5-qubit BV on IBM Mumbai on qubits 15, 17, 18, 21, 23

4-qubit BV on IBM Mumbai on qubits 15, 17, 18, 21

3-qubit BV on IBM Mumbai on qubits 15, 17, 18

Figure 6: BV outcome for 5-qubit, 4-qubit, and 3-qubit circuits.

3.1 Qubit Reuse Conditions

Condition 1. If a logical qubit □ is reused by a logical qubit □, then, there should not be any gate between □ and □.

The Condition 1 for qubit reuse is straightforward. If a logical qubit □ is to be reused by another logical qubit □, we have to make sure all gates on □ finish before all gates on □. If the two qubits have a gate, it is not possible to ensure that.

Condition 2. If a logical qubit □ is reused by a logical qubit □, then, all operations that apply to □ should not depend on any operations on □ directly or indirectly.

For example, for the DAG graph of a logical circuit in Fig. 7(a), if we reuse q1 for q4 as is shown in Fig. 7(b), gate g(q3, q1) must be finished before gate g(q4, q2), however, gate g(q3, q1) indirectly depends on gate g(q4, q2), leading to a conflict. To detect this automatically, we can see a cycle between the two groups of gates using q1 and q4. And if we insert an M gate standing for the measurement and reuse in between the two groups of gates, the cycle is also manifested. The cycle indicates that a reuse pair is invalid.

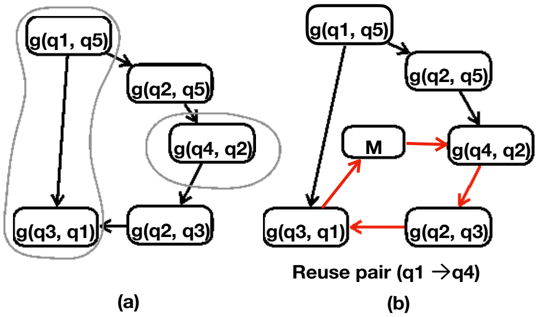


Figure 7: An invalid qubit reuse pair according to Condition 2. (a) DAG of the circuit; (b) DAG with measurement-and-reset for the (invalid) qubit reuse pair (q1 → q4).

3.2 QS-CaQR: Targeting Qubit Saving

Given a limit of qubits, we want to see if a circuit can use these limited number of qubits, and if so, provide a compiled and optimized circuit with respect to this qubit count.

We handle two different types of circuits, one without any commutable gates, and the other with commutable gates. We refer to the former as regular applications.

In both cases, we first propose how to find qubit reuse opportunities which the mid-circuit measurement and resetting could apply to. Second, since it's possible that there are many different qubit reuse opportunities, we explore the search space of qubit reuse given the same qubit limit and choose the best reuse strategy. An example of two different transformations with the same amount of qubit saving but resulting in different circuit eficiency is in Fig. 8.

3.2.1 Regular Applications.

To handle a given regular circuit with a limited number of qubits, we first construct a directed acyclic graph (DAG) that encodes the gate dependency. By analyzing the DAG graph, we can check if a qubit could be used by another. The depth of the circuit can be analyzed on the assumption that this type of reuse occurs.

We use Condition 1 and 2 to check if a qubit can be reused. With Condition 1, we check if there is any gate for the reuse pair. If Condition 1 is satisfied, we check Condition 2.

With Condition 2, we group all the gates that use qubit □, and then all the gates that use qubit □. If there are only directed edges

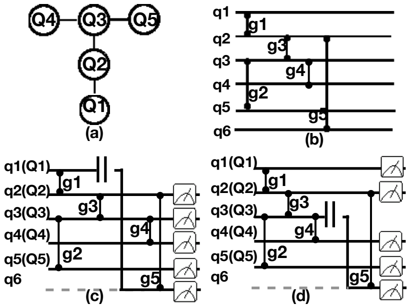


Figure 8: Using different qubits to reuse. (a) Physical architecture; (b) Original circuit (c) Compiled circuit with q1 (Q1) reused. This circuit has depth of 3. (d) Compiled circuit with q3 (Q3) reused. This circuit has depth of 4.

(transitively) from □ to □, but no directed edges from □ to □, then Condition 2 is satisfied. Otherwise, it means there is a cycle if we create such a reuse between □ and □, and it means that such a reuse pair is invalid. By applying Conditions 1 and 2, we can find all possible candidate qubit pairs (□ → □).

We evaluate one qubit pair at a time. For the qubit pair (□ → □), we add a new node □ to the DAG graph indicating a measurement-reset operation needs to be applied between the usage of □ and □. We make all gates involving □ point to □ and make node □ point to all gates involving □. The qubit reuse pair with a smaller critical path length (or circuit duration) in the corresponding DAG is better. For example, Fig. 9(a) is the DAG graph for BV circuit. To make qubit q1 be reused by qubit q3 first and make qubit q3 be reused by qubit q4 later, we added two dummy nodes D1 and D2 in the DAG graph indicating the new imposed dependency in Fig. 9(b).

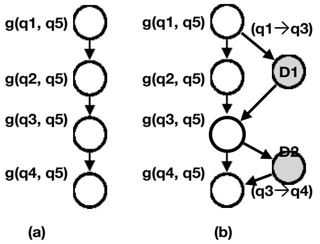


Figure 9: (a) The DAG graph of BV circuit in Fig.1(a); (b) Two added dummy nodes in the DAG graph for two-qubit reuse pairs.

Qubit Usage v.s. Circuit Duration. Qubit reuse may potentially increase circuit duration since the qubit reuse enforces the dependency between two (sets of) gates. So in the search of qubit reuse opportunity, we must carefully select qubit reuse pair (□ → □) and find the one that is less harmful to circuit depth.

Our overall strategy is to start with the original qubit count, and gradually reduce it, one at a time, until we reach the qubit count specified by the user. If the user has provided a range of qubit counts, we can generate multiple transformed versions and choose the one with the best circuit duration or fidelity (depending on the fidelity metric, for instance, estimated success probability). After logical circuit transformation, we apply a state-of-the-art qubit mapper to it. The candidate circuits for selection are the ones that are finally hardware mapped.

One qubit can be reused multiple times. Our approach allows this type of scenario flexibly, since it picks one qubit-reuse pair at one time. After one pair of reuse pair is picked, we update the circuit, and keep checking more reuse opportunities.

3.2.2 Applications with Commutable Gates. Another type of circuits is those that have non-trivial amount of gate commutativity. For instance, in the building block of the QAOA application, the CPHASE gates can commute. So there is no such gate dependency as in regular applications.

Maximal Qubit Saving. Note that for any given application, there is a bound for the maximal number of qubit saving that can be achieved. For the regular applications, we need to keep reducing qubit count and test if a limit works. But for commuting gates, due to the flexibility of reordering gates, we develop an algorithm that gives the minimal number of qubits. The only constraint for commutable-gates circuit is Condition 1 that two qubits in a reuse pair do not perform 2-qubit gate. This inspired us to use graph coloring to obtain the minimum number of qubits.

We define the qubit interaction graph □ = (□, □), where □ is the set of nodes representing the qubits in the original logical circuit, and □ is the set of edges in □ representing the gates.

Since the graph coloring algorithm requires that any two connected nodes do not share the same color. So we can apply the graph coloring algorithm on the qubit interaction graph □. The minimum number of color found means the minimum number of qubits needed. For the qubits sharing the same color, one can be reused for another, as long as all operations involving one qubit are finished before any operations involving target-reuse-qubit.

For example, we apply graph coloring algorithm to the graph with 5 vertices in Fig. 10 (a). We found out that the graph can be colored by at minimal three colors.

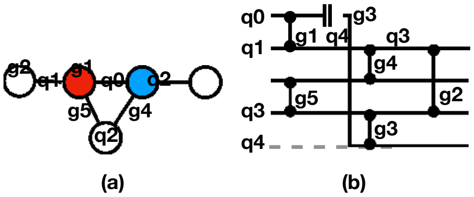


Figure 10: (a) QAOA input graph. Node q0,q2,and q4 in white color. Node q1 in blue and node q3 in red. (b) Transformed QAOA circuit with reuse qubit pair (q0→q4).

Handling Commutativity. To handle a circuit with commuting 2-qubit gates, we perform it in a similar way to the regular circuits. Firstly, we still list all valid candidate qubit pairs for reuse, (□ → □). Then we evaluate each of them individually. To generate such a candidate set, we have to apply Condition 1 and 2. Applying Condition 1 is trivial. Two qubits in a reuse pair cannot engage in the same gate. However, Condition 2 is a bit trickier. The original circuit of programs like QAOA does not have pre-determined dependence ordering between gates, hence it is difficult to test if a reuse-pair causes any circle in the execution order.

We handle Condition 2 in the following way. To make the qubit □ reuse the qubit □, we need to schedule all gates on □ first, and gates on □ later. A qubit reuse pair imposes the gate dependency between two set of gates. Hence, we let all gates on □ point to all gates on □. As we keep adding reuse pairs, there are more and more dependence edges added to the graph. Each time we add a qubit reuse-pair, we can test if it creates a cycle in the dependence graph with respect to Condition 2. If not, then such a reuse pair is a valid pair.

Algorithm for Evaluating the Impact of A Reuse. Now we describe our algorithm for evaluating the impact by applying a particular qubit-reuse pair. We maintain a list of gates in the frontier, that is, the set of gates that either do not depend on any other gate due to qubit-reuse or its dependence is resolved. Every iteration, we choose gates in the frontier to schedule, and we repeat this until no gates are left. That is, until no edges are left in □ □. Below are three steps:

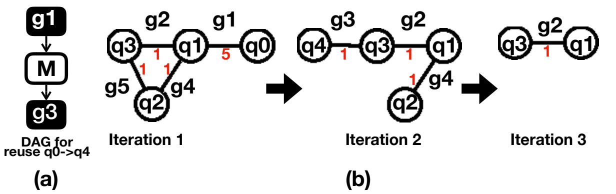


Figure 11: (a) Partial DAG graph for QAOA circuit corresponding to Fig. 10 (a) with reuse pair (q0→ q4). The M in the middle stands for a measurement and reset; (b) Gates g1 and g5 are selected at iteration 1 after perfect matching; Gates g3 and g4 are selected at iteration 2; and gate g2 is selected at iteration 3. The weight of edge is in red. Transformed circuit in Fig. 10

Step 1: We use the pair (□ → □) to update the current dependence graph □, or to create dependence edges if it is the first qubit-reuse pair to apply. The qubit reuse pair imposes the dependency on the gates related to □ and □. We add a new node that represents the measurement-and-reset operation between two sets of gates. That is, all gates contains □ should point to the new gate, and all gates contains □ should be pointed to by this new gate.

Step 2: We temporarily remove all gates that correspond to the gates with none-zero in-degree in the dependence graph □ from the qubit interaction graph □ □. The reason for this is that those gates rely on other gates and cannot be scheduled until their dependencies are resolved.

Now we assign weights to the edges in □ □. We want to prioritize the gates that have gates depending on it – the type of gates involving qubits to be reused. So we assign larger weights to these gates as a parameter |□ □| > 1, while all other edges' weight is 1.

Step 3: We apply maximum weight perfect matching algorithm to □ □. Those gates with higher priority would be more likely selected. The maximal matching algorithm will select as many gates as possible to improve the parallelism. Those selected gates are scheduled and the corresponding edges in □ □ □and nodes in □ are removed permanently. Then we place the temporarily removed gates back to □ □. We now go back to Step 2 until the frontier has no gate.

With the above three steps, we can get the duration of the circuit after applying the reuse pair. Since we have tried to maximize the parallelism by perfect matching, we got a circuit that has good duration. We associate the duration with such a qubit reuse pair. An illustration of the complete QAOA circuit transformation process is depicted in Fig. 11.

Similarly, for this type of applications, our compiler takes a circuit as input, and a limit of qubits. It outputs two types of results: (1) Yes, there exists a way to build the circuit with this limit on qubits, or (2) no. For each given qubit limit, we process it in the same way as that for regular applications. We evaluate and rank different qubit-reuse pairs, and choose the best one with respect to circuit duration/fidelity. At each time, we save one qubit. Then we keep saving until we reach the given qubit limit, or until no more qubit can be saved. If we are given a range of qubit-counts, we will be able to generate different versions of logical circuits that can be used for further selection with respect to user requirement.

3.3 SR-CaQR: Targeting Reduction of SWAPs and Improved Fidelity

Now assuming we have enough resources. We design SR-CaQR to compile a given circuit and treat the SWAP gate reduction as the primary goal through qubit reuse.

The main reason we can achieve SWAP gate reduction is because we can delay the first gate for some qubits without extending the critical path. Those qubits that have not started any operations can map to two type of physical qubits. One is the fresh physical qubit that is not used by any logical qubits. The another one is the used physical qubit but which has done all operations on that. With a broader selection of physical qubits, we can pick the one (or two) with smaller distance (or with lower error rate) in architecture coupling when one (or two) new logical qubit has to be mapped such that the SWAP gates are reduced. Since the gate delay would not extend the critical path of circuit but SWAP gates are reduced, the depth of compiled could be potentially reduced as well.

SR-CaQR considers the solution for both application types as QS-CaQR. The details of the design are in the following.

3.3.1 Regular Applications.

For regular program compilation, we take the logical circuit and the hardware information as the inputs. Then, we compile the circuit layer by layer and map the logical qubit to physical quit when necessary. The new logical qubit will pick the best available physical qubit to minimize the SWAP gates and improve fidelity.

Step1: We construct the DAG graph □ for the input circuit and maintain a list of physical qubits that are available to use in the list □ □ □ □ □ □ □. Initially, □□ □ □ □ □ □ □contains all physical qubits. By doing the analysis on DAG graph, we can easily find out that whether a gate is in critical path or not.

Step 2: Considering the gate with qubit(s) not mapped. For all gates with in-degree = 0 in the frontier of DAG, if the gate is not on the critical path, we delay it. If the gate is on the critical path, we must run it. It it possible that this gate has both qubits not mapped, then we map the qubit with more gates on it first. This logical qubit will pick a physical qubit from □□□□□□□that□also□can benefit

future gates involving it (by lookahead). Or it will map to a physical qubit with better connectivity. The mapped physical qubit is then removed from □□□□□□.□□□□□

For another unmapped logical qubit of the gate with only one qubit already mapped, we pick the one from □□□□□□□□□ with minimum distance to its already mapped qubit. If there is a tie, we pick the qubit with smaller readout error or the qubit connected by a physical link with smaller CNOT error. The mapped physical qubit is removed from □□□□□□.□□□□□

Step 3: Considering the gate with both qubits mapped. For all gates in the frontier of DAG whose both qubits are mapped, if the gate is hardware-compliant, we schedule it. If two qubits are not adjacent to each other, we add SWAP gates. We use a heuristic method to insert SWAP gate with the consideration of error variability and the side-effect on the following gates.

Step 4: If any gates are scheduled, we update the frontier in DAG graph □□. If there is any qubit done with all operations on it, we added this qubit back to □□□□□□□. We□repeat□the step 2-4 until the frontier is empty.

We use an example to explain our method. Fig. 12 (a) is a logical circuit with the physical coupling graph on the top of it. There is nothing mapped on the coupling graph. In Fig. 12 (b), only two qubits q4 and q0 are mapped and g2 are scheduled. This is because gate g1 is not on the critical path which is delayed and g2 is on the critical path. q4 is mapped first since more gates apply on it and is mapped to the middle of the architecture coupling graph.

In Fig. 12 (c), after gate g2 is scheduled, we reclaim the qubit q0 and save it to □□□□□□□since□there□is no more gates on qubit q0. Now gate g1 and g3 are both on the critical path. For gate g1, we map q1 first since it involves more gates. Qubit q1 will interact with q4 later, so we map it close to q4. For gate g3, q4 is already mapped, then we map q3 close to it.

In Fig. 12 (d), we free q3 and q2 since they are done with all operations. Gate g4 has two mapped qubits and is hardware-compliant.

In this example, SR-CaQR added zero SWAP gate by applying qubit reuse. However, the original circuit at least takes one SWAP gate to make the circuit hardware-compliant.

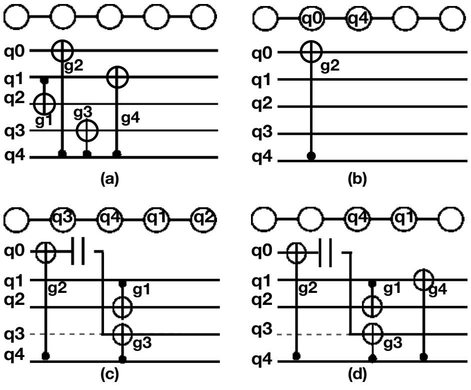


Figure 12: Example of SR-CaQR for regular applications

3.3.2 Applications with Commuting Gates. SR-CaQR also considers the solution for compiling applications with commutable gates such as QAOA. The main idea is similar to that for regular applications. We try to delay the start time of some qubits such that those qubit would have more options of candidate physical qubits. However, unlike the regular applications, applications with commutable gates do not have gate dependency. To solve this problem, we can utilize □□QS-CaQR to manually impose gate dependency for a part of the gates. The details are in the following.

Step 1: Constructing a partial DAG □□. For the given QAOA circuit, we use QS-CaQR to find a sweet point of number of qubit reuse and the corresponding qubit reuse pairs (□□ → □□). Then, we construct a partial DAG based on the qubit reuse pairs. All gates that apply to qubit □□

depend on all gates that apply to qubit □□. The DAG □□ also contains gates that do not involve the qubit reuse. Those gates have in-degree = 0. After graph □□ constructed, we update the degree information since the imposed qubit reuse and gate dependency increase the degree of □□. Then, we start scheduling those gates with in-degree = 0.

Step 2: Delaying gates. For the gate with one qubit or two qubits not mapped, we want to delay it based on the following two conditions. Firstly, if the gate is in the reuse dependency graph, we do not delay it. Since all gates in the reuse dependency have to be scheduled first such that the corresponding qubit can be measured and reused by other reuse-target qubits. Secondly, if the gate has one or two qubits with high degree on QAOA graph, we do not delay it. The highest degree in the QAOA graph determines the lower bound of depth of compiled circuit, so delaying gate with high degree qubit would potentially increase the circuit depth.

Step 3: Scheduling gates. If the gate is not delayed or has both qubit mapped, we schedule it. For the gate with both qubits mapped, we schedule it if it is hardware-compliant. Otherwise, we add SWAP gates for it heuristically. For the gate with one or two qubits unmapped (but not delayed), we map its logic qubits with the consideration of qubit distance and error variability which is the same as the regular application solution. Then, we schedule it or add SWAP gates for it.

Step 4: Update information and repeat. We remove those scheduled gates from □□ and update the degree information. Here, we still reclaim qubits if they are done with all their operations. Then we repeat step 2-4 until □□ is empty.

3.4 Overhead Analysis

We consider two methods QS-CaQR and SR-CaQR, both having a time complexity for general circuits expressed as O(□ □^{3}), where k is the number of qubits and n is the number of gates in the circuit.

This main time complexity arises from the iterative process of checking Condition 2 for each qubit-pair in the circuit. We track the direct/indirect dependence between each pair of gates, which is O(□^{2}). Such dependence information also implicitly gives us the valid qubit-reuse pairs. For each qubit-reuse pair, we need to calculate the resulted circuit critical path, which is O(n). In the outermost loop, it takes at most □ iterations, since at least 1 qubit is reduced once. Hence, the worst-case time complexity is O(k□^{3}). For both methods, the general circuit is similar in structure and thus has the same time complexity.

For QAOA benchmarks, the major overhead of the time complexity comes from finding the maximum matching, which we accomplish using the Edmonds' Blossom algorithm. This algorithm has a time complexity of $O(\square^3)$, where n is the number of gates in the circuit (which also means the number of edges in the problem graph). This occurs for each qubit-pair under consideration. Therefore, the worst case time complexity for QAOA benchmarks in both methods becomes $O(\square^3\square^4)$. In practice, we do not actually incur the worst case overhead. Moreover, note that Edmond's algorithm finds optimal matching. We can replace it with the standard greedy algorithm for computing a maximal matching, which is more efficient and in practice computes a matching that is very close to optimal. We leave this as our future work.

4 EVALUATION

In this section, we evaluate our proposed methods by using different types of quantum circuits: regular circuits without commutable gates, and circuits with commutable gates. We explore qubit reuse opportunities in a given circuit and analyze the trade-off between qubit usage, circuit depth/duration, and gate count. We also perform real machine experiments to show that qubit reuse could help improve end-to-end circuit fidelity and performance.

4.1 Experiment Setup

Architectures and Backend. Both QS-CaQR and SR-CaQR are using IBM heavy-hex as the backends. When the qubit number is large, we use the scaled heavy-hex architecture.

For real machines, we use IBM Mumbai which also exhibits a heavy-hex pattern. We also use the real calibration data exported from the IBM systems including the CNOT duration, CNOT error for each physical link, and qubit readout errors.

Metrics. To assess our proposed method, we use qubit usage, two-qubit gate count, and circuit depth/duration as metrics. We use total variant distance (TVD) if necessary. We also use the final application outcome on real machines as a way to evaluate the effectiveness of our method.

Qubit reuse enforces extra gate dependency in the circuit and potentially increases the circuit duration. A circuit with a smaller duration has less decoherence error. So we need to evaluate circuit duration. Two-qubit gate count is another concern. Qubit reuse could potentially save the number of SWAP gates inserted. Since we can reuse a nearby qubit if both conditions in Section 3.1 are satisfied to reduce the communication cost for a gate involving two distant qubits.

Baselines. We use IBM Qiskit as the baseline, with optimization level 3 turned on. It compiles both regular and commutable-gate circuits. QAOA is a classical-quantum application. Our optimization is on the quantum part. It also needs a classical optimizer. For running QAOA for the full experiment, we use the well-known "COBYLA" classical optimizer provided by IBM Qiskit by default.

Benchmarks. We have two types of benchmarks, regular quantum applications, and commutable-gate quantum applications. We have the regular quantum applications: Rd-32, 4mod5, Multiply_13, System_19, CC_10, XOR_5, and BV_10 [4][14]. For the commutable-gate circuits, we use QAOA circuits for the max-cut problem. We have two different types of input problem graphs, random graph and power-law graph, with different sizes, from 16 to 128.

4.2 QS-CaQR Evaluation

In this section, we evaluate the qubit-saving version: QS-CaQR. QS-CaQR first applies qubit reuse on logical circuits.

For each application, we tried different qubit limit numbers, and generate different compiled circuits. For each circuit corresponding to a desired qubit usage, we use the Qiskit transpiler to insert SWAPs. We show the results for both regular applications and QAOA applications.

4.2.1 Regular Applications. We show the results of three regular applications, Multiply_13, System_9, and BV_10 in Fig. 13. The bars on the right half of Fig. 13 (a), (b), and (c) represent the depth of logical circuits with respect to qubit usage reduction, and the grey bar plots on the left half figures represent the depth of final compiled circuits with hardware mapping. We can see that, for all logical circuits, when the number of qubit usage decreases (from right to left in the x-axis), the circuit depth increases. However, the results for the final circuit show a different pattern. When the qubit number decreases, the depth decreases slowly in the beginning and increases in the end. It is potentially because when qubit-saving is too aggressive, it does not help SWAP insertion.

Depending on user request, we can choose different compiled circuits. If the users demand to have a minimal depth circuit, we need to choose a version that saves some qubits moderately. The sweet spot is usually in the middle. If the users demand to save qubits, it will not have the best depth.

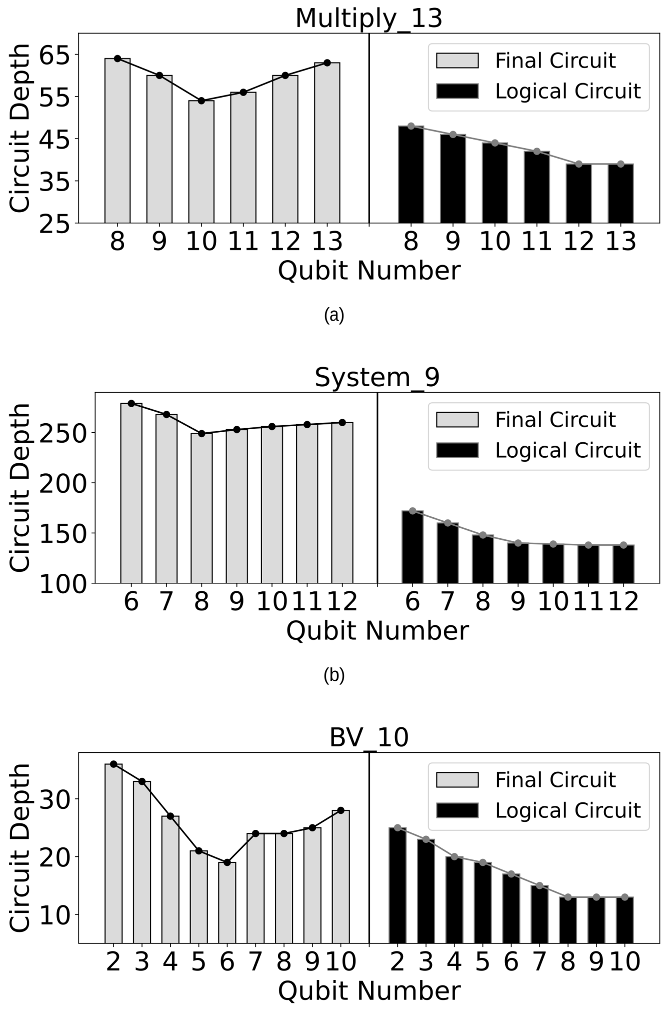
4.2.2 QAOA Applications. We also explore the trade-off between circuit depth and qubit usage in QAOA circuits. We did experiments on random and power-law graphs with the number of vertices of 16, 32, 64, and 128. Each of the graphs has a density of 30%. We show results in Fig. 14. The result of graphs with 64 vertices are already shown in the motivation section, we only present results of 16, 32, and 128 qubits here.

It turns out that the QAOA circuit has more qubit reuse opportunities than regular quantum applications, especially for large cases. The minimum qubit usage of the power-law graph is closer to zero. For both random graphs and power-law graphs, QAOA can save at least half of the qubits in the extreme case.

Compared with the random graphs, the power-law graphs have more reuse and a better tradeoff between depth and qubit number. This makes sense since the power-law graph contains more vertices with low degrees and the corresponding qubits have fewer gates on them. And the large degree node dominates the overall depth. This makes those qubits could be reused easily without sacrificing too much circuit depth.

4.2.3 Tradeoff Analysis. Since we generated different versions of circuits with respect to different qubit numbers for the same application, we can perform a tradeoff analysis shown in Table. 1.

We compare the version that has maximal reuse, the version that has minimal depth, and the baseline of Qiskit with the highest optimization level. It shows that if targeting maximal reuse, then the circuit depth and duration will be affected adversely. If targeting minimal depth using the QS-CaQR version, we have moderate qubit

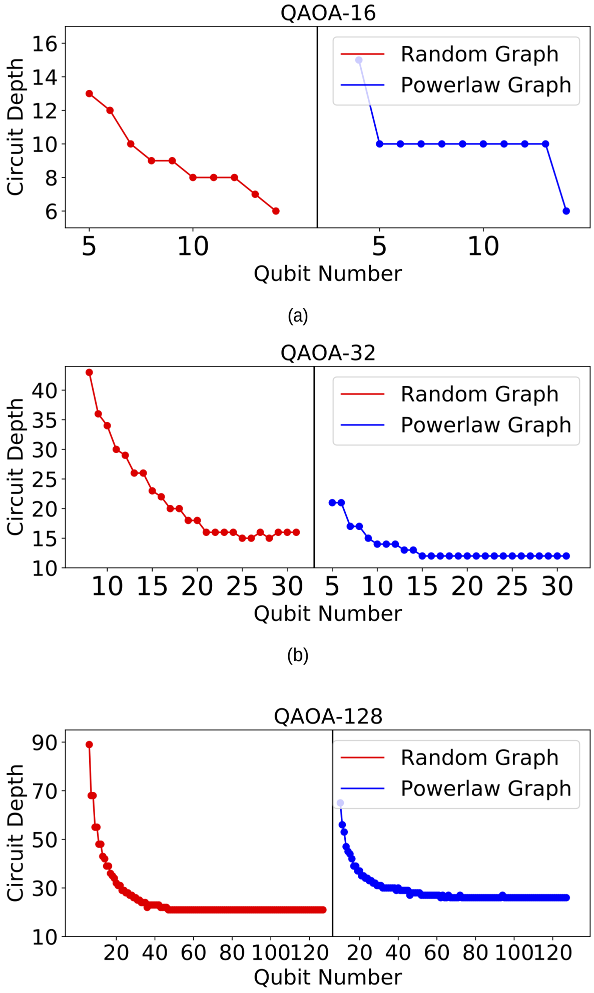


(a)

(b)

(c)

Figure 13: QS-CaQR: Reuse vs depth for regular circuits.



(a)

(b)

(c)

Figure 14: Reuse vs depth for QAOA circuit (logical circuits)

saving. But for both versions of QS-CaQR, we are better than the baseline surprisingly for circuit depth/duration in a lot of cases. This demonstrates the usefulness of qubit reuse has extended beyond qubit saving.

4.3 SR-CaQR Evaluation

To evaluate the performance of SR-CaQR, we compare it with QS-CaQR. Firstly, we use SR-CaQR to compile the given circuit. Then for fairness, we use the version in QS-CaQR that has a minimal SWAP number (we exhaust all possible qubit-saving count). Both experiments are conducted on IBM Mumbai's architecture. We show results in Table 2.

4.3.1 Regular Applications. For all regular applications, SR-CaQR has the same or better SWAP gate count. For the 4mod5 circuit, SR-CaQR minimizes the SWAP gate count to zero. For System_9 circuit, SR-CaQR has 20.5% of SWAP gate reduction.

4.3.2 QAOA Applications. For the QAOA applications, SR-CaQR has a similar SWAP number compared with QS-CaQR for small applications since the near-optimal compilation is achieved by both solutions. For larger input graph size which has nodes larger than 15, the SR-CaQR uses fewer SWAPs, and the duration time is also reduced. This demonstrates the usefulness of SR-CaQR.

4.4 Real Machine Experiments

We run BV_5, BV_10, Multiply_13, CC_13 circuit, QAOA 10-0.3, QAOA 10-0.5 circuit at IBM Mumbai device. This is the one machine supporting dynamic circuits. Not all IBM machine support that. For regular circuits, we use TVD to evaluate the output distribution. The results are shown in Table 3. A takeaway is that our SR-CaQR has improved for all the benchmarks listed here. Since the current real machine is still in an early stage and the mid-measurement pulse is

Table 1: QS-CaQR Version: The unit of circuit duration is dt – system cycles. 1 dt is 0.22 nano-seconds.

	Baseline (No Reuse)				Ours with Maximal Reuse				Ours with Minimal Depth			
Benchmarks	Qubit	Depth	Duration	SWAP	Qubit	Depth	Duration	SWAP	Qubit	Depth	Duration	SWAP
RD-32	4	28	88.6k	9	3	24	71.6K	6	3	24	71.6K	6
4mod5	5	20	81.1K	5	4	18	61.5K	3	4	18	61.5K	3
Multiply_13	13	63	129K	35	8	58	145K	17	11	54	91.5K	26
System_9	12	279	431K	95	7	249	401K	45	6	230	314K	39
BV_10	10	30	92.3K	18	2	28	144K	0	7	23	72.4K	5
CC_10	10	15	88.7K	9	2	13	122K	0	5	10	72.6K	1
XOR_5	6	6	45.5K	5	2	7	42.7K	0	4	4	42.7K	0
QAOA5-0.3	5	8	44.2K	3	3	5	34.5K	1	3	5	21.9K	1
QAOA10-0.3	10	15	77K	13	7	11	65.5K	4	7	11	51.5K	4
QAOA15-0.3	15	41	164K	38	4	48	207K	0	10	32	102K	11
QAOA20-0.3	20	64	282K	107	4	72	509K	0	16	37	201K	59
QAOA25-0.3	25	123	561K	172	5	133	820K	0	22	65	391K	154

Table 2: SR-CaQR (MIN-SWAP) v.s. QS-CaQR

	QS-CaQR (MIN-SWAP)				SR-CaQR			
Benchmarks	Qbt.	Dpth.	Drt.	SWP	Qbt.	Dpth.	Drt.	SWP
RD-32	3	24	71.6K	6	3	24	71.6K	6
4mod5	4	15	59.5K	0	4	15	59.5K	0
Multiply_13	11	54	91.5K	26	11	54	84.2K	23
System_9	6	230	314K	39	10	234	269K	31
BV_10	7	24	69.2K	3	7	24	69.2K	3
CC_10	6	9	70.2K	2	6	9	70.2K	2
XOR_5	4	0	42.7K	0	4	0	42.7K	0
QAOA5-0.3	3	5	21.9K	1	3	5	21.9K	1
QAOA10-0.3	7	11	51.5K	4	6	10	48.2K	2
QAOA15-0.3	8	32	102K	8	8	27	98.2K	5
QAOA20-0.3	15	37	201K	52	15	35	195K	51
QAOA25-0.3	20	65	391K	124	20	63	360K	114

Table 3: TVD results for BV, Multiply, CC circuit

Benchmarks	TVD (Baseline)	TVD (SR-CaQR)
Multiply_13	0.76	0.61
BV_10	0.64	0.48
CC_10	0.61	0.44

not stable, so we hope to see results for all these benchmarks once the machine supporting dynamic circuit becomes more mature.

For QAOA circuits, the results are shown in Fig. 15 and Fig. 16. For both figures, the x-axis stands for the round number of the parameter optimization by a classical machine learning optimizer called COBYLA. The y-axis is the negation of the expected value of the max-cut value. The smaller is better. The red curve is the result of SR-CaQR with 6 qubits and the blue curve is the result of the circuit without qubit reuse. For both experiments, SR-CaQR circuits achieve better max-cut values and converge faster.

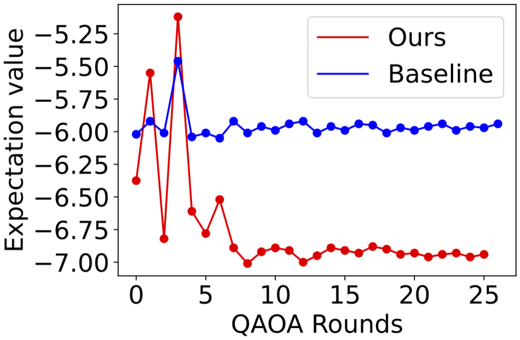


Figure 15: The end-to-end result of the QAOA reuse experiments for QAOA-10 with density 30%

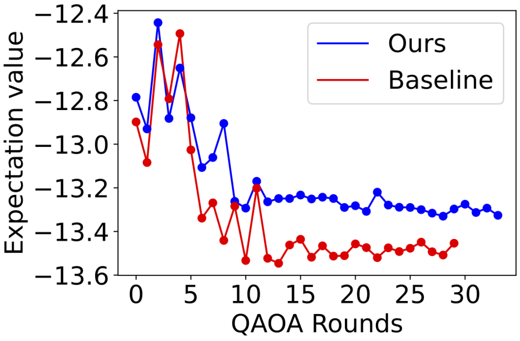


Figure 16: The end-to-end result of the QAOA reuse experiments for QAOA-10 with density 50%.

5 RELATED WORK

Generic compilers, such as [15, 17, 22, 24–26, 29–33] compile the given quantum circuit with the consideration of circuit depth, gate count, and qubits variability. After the applications with gate commutativity such as QAOA [6–8] and VQE [20] gained more attention, domain-specific compilers exploit the commutativity feature. Some new compilers exploiting the such feature are proposed [1, 13, 16]. But none of those are aware of the opportunities in qubit reuse.

Paler et al. [19] proposed a method, named wire recycling, for quantum circuit synthesis with the same number of qubit but less wire. They construct a causal graph from the original circuit and utilize the graph search algorithm to find such qubit wire recycling opportunities. Even the wire recycling only applies to the ancilla qubit in the quantum reversible circuit, their method could be adapted to find the qubit reuse opportunity in QAOA. In addition, they also provide a table of reversible circuits that proves the qubit reuse is pragmatic. But in their scenarios, the ancilla qubits are pre-determined prior to use

SQAURE[5] is a qubit reuse compilation framework built upon the un-computation strategy. It has a locality-aware allocation strategy to decide which ancilla qubit to be reused to reduce the circuit communication overhead. It also has a cost-effective reclamation strategy to decide where the un-computation should be applied. However, this compilation framework leveraging the un-computation can only be applied to reversible arithmetic circuits and reclaim ancilla qubits. In addition, the un-computation strategy inevitably introduces more gates to reset ancilla qubit back to state |0>. As a result, it makes compiled circuit have a longer circuit depth and larger accumulated gate errors. Note that the reclamation through un-computation is limited to ancilla qubit only. In our case, we still need the outcome result of the qubits we measure, before reusing them. Ancilla qubits do not need to be measured.

Govia et al. [9] proposed a randomized benchmark suite for mid-circuit measurements. It can be used to test the impact of mid-circuit measurement. This work is meaningful in device characterization and is complementary to our work.

6 CONCLUSION

With supported mid-circuit hardware measurement, we can improve circuit efficacy and fidelity from three aspects: (a) reduced qubit usage, (b) reduced swap insertion, and (c) improved estimated success probability. We demonstrate this using real-world applications Bernstein Verizani on real hardware and show that circuit resource usage can be improved by 60%, and circuit fidelity can be improved by 15%. We design a compiler-assisted tool that can find and exploit the tradeoff between qubit reuse, fidelity, gate count, and circuit duration.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their constructive and helpful feedback. We thank Zhiding Liang and Jinglei Chen for providing valuable comments during discussion. This work is supported by grants from Rutgers Research Council and NSF-FET-2129872. It is also partially supported by the U.S. Department of Energy, Office of Science, National Quantum Information Science Research Centers, Co-design Center for Quantum Advantage (C2QA) under contract number DE-SC0012704. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

REFERENCES

[1] Mahabubul Alam, Abdullah Ash-Saki, and Swaroop Ghosh. 2020. Circuit Compilation Methodologies for Quantum Approximate Optimization Algorithm. In 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). 215–228. https://doi.org/10.1109/MICRO50266.2020.00029

[2] Christopher Chamberland, Guanyu Zhu, Theodore J Yoder, Jared B Hertzberg, and Andrew W Cross. 2020. Topological and Subsystem Codes on Low-Degree Graphs with Flag Qubits. Phys. Rev. X 10, 1 (jan 2020), 11022. https://doi.org/10.1103/PhysRevX.10.011022

[3] A. D. Córcoles, Maika Takita, Ken Inoue, Scott Lekuch, Zlatko K. Minev, Jerry M. Chow, and Jay M. Gambetta. 2021. Exploiting Dynamic Quantum Circuits in a Quantum Algorithm with Superconducting Qubits. Phys. Rev. Lett. 127 (Aug 2021), 100501. Issue 10. https://doi.org/10.1103/PhysRevLett.127.100501

[4] Andrew Cross, Ali Javadi-Abhari, Thomas Alexander, Niel De Beaudrap, Lev S Bishop, Steven Heidel, Colm A Ryan, Prasahnt Sivarajah, John Smolin, Jay M Gambetta, et al .2022. OpenQASM 3: A broader and deeper quantum assembly language. ACM Transactions on Quantum Computing 3, 3 (2022), 1–50.

[5] Yongshan Ding, Xin-Chuan Wu, Adam Holmes, Ash Wiseth, Diana Franklin, Margaret Martonosi, and Frederic T. Chong. 2020. SQUARE: Strategic Quantum Ancilla Reuse for Modular Quantum Programs via Cost-Effective Uncomputation. In 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA). IEEE. https://doi.org/10.1109/isca45697.2020.00054

[6] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. 2014. A Quantum Approximate Optimization Algorithm. arXiv:1411.4028 [quant-ph]

[7] E. Farhi, J. Goldstone, S. Gutmann, and H. Neven. 2017. Quantum Algorithms for Fixed Qubit Architectures. arXiv:1703.06199 [quant-ph]

[8] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Leo Zhou. 2021. The Quantum Approximate Optimization Algorithm and the Sherrington-Kirkpatrick Model at Infinite Size. arXiv:1910.08187 [quant-ph]

[9] L. C. G. Govia, P. Jurcevic, S. T. Merkel, and D. C. McKay. 2022. A randomized benchmarking suite for mid-circuit measurements. https://doi.org/10.48550/ARXIV.2207.04836

[10] Lov K Grover. 1996. A fast quantum mechanical algorithm for database search. In Proceedings of the twenty-eighth annual ACM symposium on Theory of computing. 212–219.

[11] IBM. 2022. Introduction to Dynamic Circuits. https://quantum-computing.ibm.com/lab/docs/iql/manage/systems/dynamic-circuits/02-Introduction-To-Dynamic-Circuits.

[12] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M Chow, and Jay M Gambetta. 2017. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. Nature 549, 7671 (2017), 242–246.

[13] Lingling Lao and Dan E. Browne. 2022. 2QAN: A Quantum Compiler for 2-Local Qubit Hamiltonian Simulation Algorithms. In Proceedings of the 49th Annual International Symposium on Computer Architecture (New York, New York) (ISCA '22). Association for Computing Machinery, New York, NY, USA, 351–365. https: //doi.org/10.1145/3470496.3527394

[14] Ang Li, Samuel Stein, Sriram Krishnamoorthy, and James Ang. 2021. QASMBench: A Low-level QASM Benchmark Suite for NISQ Evaluation and Simulation. arXiv preprint arXiv:2005.13018 (2021).

[15] Gushu Li, Yufei Ding, and Yuan Xie. 2019. Tackling the qubit mapping problem for NISQ-era quantum devices. In Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. ACM, 1001–1014.

[16] Gushu Li, Anbang Wu, Yunong Shi, Ali Javadi-Abhari, Yufei Ding, and Yuan Xie. 2022. Paulihedral: A Generalized Block-Wise Compiler Optimization Framework for Quantum Simulation Kernels. In Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (Lausanne, Switzerland) (ASPLOS 2022). Association for Computing Machinery, New York, NY, USA, 554–569. https://doi.org/10.1145/3503222.3507715

[17] Prakash Murali, Jonathan M. Baker, Ali Javadi-Abhari, Frederic T. Chong, and Margaret Martonosi. 2019. Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale Quantum Computers. In Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (Providence, RI, USA) (ASPLOS '19). ACM, New York, NY, USA, 1015–1029. https://doi.org/10.1145/3297858.3304075

[18] Adam Paetznick and Krysta M. Svore. 2013. Repeat-Until-Success: Non-deterministic decomposition of single-qubit unitaries. (2013). https://doi.org/10.48550/ARXIV.1311.1074

[19] Alexandru Paler, Robert Wille, and Simon J. Devitt. 2016. Wire recycling for quantum circuit optimization. Physical Review A 94, 4 (oct 2016). https://doi.org/10.1103/physreva.94.042337

[20] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. 2014. A variational eigenvalue solver on a photonic quantum processor. Nature Communications 5, 1 (jul 2014). https://doi.org/10.1038/ncomms5213

[21] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L O'brien. 2014. A variational eigenvalue solver on a photonic quantum processor. Nature communications 5 (2014), 4213.

[22] Alireza Shafaei, Mehdi Saeedi, and Massoud Pedram. 2014. Qubit placement to minimize communication overhead in 2D quantum architectures. In 2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE, 495–500.

[23] Peter W Shor. 1999. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM review 41, 2 (1999), 303–332.

[24] Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Caroline Collange, and Fernando Magno Quintão Pereira. 2019. Qubit Allocation as a Combination of Subgraph Isomorphism and Token Swapping. Proc. ACM Program. Lang. 3, OOPSLA, Article 120 (Oct. 2019), 29 pages. https://doi.org/10.1145/3360546

[25] Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Sylvain Collange, and Fernando Magno Quintão Pereira. 2018. Qubit allocation. In Proceedings of the 2018 International Symposium on Code Generation and Optimization. ACM, 113–125.

[26] Bochen Tan and Jason Cong. 2020. Optimal Layout Synthesis for Quantum Computing. In Proceedings of the 39th International Conference on Computer-Aided

Design (Virtual Event, USA) (ICCAD '20). Association for Computing Machinery, New York, NY, USA, Article 137, 9 pages. https://doi.org/10.1145/3400302.3415620

[27] Wei Tang, Teague Tomesh, Martin Suchara, Jeffrey Larson, and Margaret Martonosi. 2021. CutQC: Using Small Quantum Computers for Large Quantum Circuit Evaluations. In Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (Virtual, USA) (ASPLOS '21). Association for Computing Machinery, New York, NY, USA, 473–486. https://doi.org/10.1145/3445814.3446758

[28] Swamit S Tannu and Moinuddin K Qureshi. 2019. Mitigating Measurement Errors in Quantum Computers by Exploiting State-Dependent Bias. In Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '52). Association for Computing Machinery, New York, NY, USA, 279– 290. https://doi.org/10.1145/3352460.3358265

[29] Swamit S. Tannu and Moinuddin K. Qureshi. 2019. Not All Qubits Are Created Equal: A Case for Variability-Aware Policies for NISQ-Era Quantum Computers. In Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (Providence, RI, USA) (ASPLOS '19). ACM, New York, NY, USA, 987–999. https://doi.org/10.1145/3297858.3304007

[30] Robert Wille, Lukas Burgholzer, and Alwin Zulehner. 2019. Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations. In Proceedings of the 56th Annual Design Automation Conference 2019. ACM, 142.

[31] Chi Zhang, Ari B. Hayes, Longfei Qiu, Yuwei Jin, Yanhao Chen, and Eddy Z. Zhang. 2021. Time-Optimal Qubit Mapping (ASPLOS 2021). Association for Computing Machinery, New York, NY, USA, 360–374. https://doi.org/10.1145/3445814.3446706

[32] Alwin Zulehner, Stefan Gasser, and Robert Wille. 2017. Exact Global Reordering for Nearest Neighbor Quantum Circuits Using A*. In International Conference on Reversible Computation. Springer, 185–201.

[33] Alwin Zulehner, Alexandru Paler, and Robert Wille. 2018. Eficient mapping of quantum circuits to the IBM QX architectures. In 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 1135–1138.

Received 2022-10-20; accepted 2023-01-19